**Mathematical Framework for Tick-Based Strategy Validation**

**1. Tick Data Statistical Foundation**

**Microstructure Return Analysis**

python

*# Tick-level returns*

tick\_returns = []

for trade in trades:

entry\_mid = (trade.entry\_bid + trade.entry\_ask) / 2

exit\_mid = (trade.exit\_bid + trade.exit\_ask) / 2

tick\_return = (exit\_mid - entry\_mid) / entry\_mid

tick\_returns.append(tick\_return)

*# Account for discrete price grid*

price\_improvement = (fill\_price - limit\_price) / tick\_size

execution\_alpha = price\_improvement \* tick\_value

*# Test for microstructure noise*

Hasbrouck\_Information\_Share = var(efficient\_price) / var(transaction\_price)

if Hasbrouck\_IS < 0.8:

warning("High microstructure noise")

**Edge Quantification (Tick-Based)**

python

*# Expected Value per Trade (in ticks)*

E[Ticks] = win\_rate \* avg\_win\_ticks - loss\_rate \* avg\_loss\_ticks

*# Adjust for market conditions*

spread\_cost = avg\_spread / 2 *# Half spread per side*

tick\_slippage = 0.3 *# Empirical slippage*

commission\_ticks = commission / tick\_value

*# Net Edge*

Net\_Edge = E[Ticks] - spread\_cost - tick\_slippage - commission\_ticks

Required: Net\_Edge > 0.5 ticks

*# Confidence bounds using tick clusters*

tick\_outcomes = [1, 1, 2, -1, -2, 3, ...] *# Discrete outcomes*

bootstrap\_CI = percentile(bootstrap\_means, [2.5, 97.5])

**2. Regime-Dependent Performance**

**Markov Regime Switching Model**

python

*# Define states*

S = {Trending, Ranging, Volatile}

*# Transition probability matrix*

P = [p\_ij] where p\_ij = P(S\_t+1 = j | S\_t = i)

*# Estimate from data:*

p\_ij = count(transitions from i to j) / count(state i)

*# Strategy performance by regime*

E[R|S] = expected return given state S

σ[R|S] = volatility given state S

*# Optimal strategy selection*

π\*(S) = argmax\_strategy E[R|S] / σ[R|S] *# Maximum Sharpe by regime*

**Regime Detection Accuracy**

python

*# Confusion matrix for regime classification*

Accuracy = (TP + TN) / Total

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

*# F1 Score for regime detection*

F1 = 2 \* (Precision \* Recall) / (Precision + Recall)

Target: F1 > 0.7 for each regime

**3. Risk-Adjusted Performance Metrics**

**Modified Sharpe Ratio (for non-normal returns)**

python

*# Cornish-Fisher adjustment*

SR\_modified = SR\_standard \* (1 + S/6 \* SR\_standard - (K-3)/24 \* SR\_standard²)

*# Probabilistic Sharpe Ratio*

PSR = Φ((SR\_observed - SR\_benchmark) \* √n / √(1 - S\*SR + (K-3)/4 \* SR²))

where Φ = cumulative standard normal

**Maximum Drawdown Distribution**

python

*# For strategy validation, calculate:*

MDD\_expected = σ \* √(2 \* log(T)) *# Theoretical maximum*

MDD\_observed = max peak-to-trough decline

*# Calmar Ratio*

Calmar = Annual\_Return / MDD\_observed

Target: Calmar > 1.0

**Risk of Ruin**

python

*# Kelly Criterion for position sizing*

f\* = (p \* b - q) / b

where:

p = win probability

b = win/loss ratio

q = 1 - p

*# Probability of 50% drawdown*

P(ruin) = ((1-f\*p)/(1+f\*b))^(C/2)

where C = initial capital in units

Target: P(ruin) < 0.01

**4. Robustness Testing**

**Parameter Stability Analysis**

python

*# For each parameter θ:*

Performance(θ) = Sharpe ratio at parameter value θ

*# Stability metric*

Stability = 1 - std(Performance(θ ± 20%)) / mean(Performance(θ))

Target: Stability > 0.8

*# Parameter sensitivity*

∂Performance/∂θ = gradient of performance

Low sensitivity = |gradient| < 0.1

**Monte Carlo Validation**

python

*# Bootstrap resampling*

for i in 1:10000:

sample = random.choice(trades, size=n, replace=True)

metrics[i] = calculate\_performance(sample)

*# Confidence intervals*

CI\_lower = percentile(metrics, 2.5)

CI\_upper = percentile(metrics, 97.5)

*# Probability of positive returns*

P(profit) = sum(metrics > 0) / 10000

Target: P(profit) > 0.95

**Walk-Forward Efficiency**

python

*# For each window:*

IS\_performance = in\_sample\_sharpe

OOS\_performance = out\_of\_sample\_sharpe

*# Walk-forward efficiency*

WFE = mean(OOS\_performance) / mean(IS\_performance)

Target: WFE > 0.5 *# At least 50% of IS performance*

*# Consistency*

Consistency = count(OOS\_performance > 0) / total\_windows

Target: Consistency > 0.7

**5. Market Impact and Capacity**

**Slippage Model**

python

*# Linear impact model*

Slippage = α + β \* √(Order\_Size / ADV)

where ADV = Average Daily Volume

*# Estimate from data:*

actual\_price = fill\_price - mid\_price\_at\_signal

Run regression: actual\_price ~ order\_size

*# Capacity calculation*

Max\_Size = ADV \* 0.01 *# 1% of daily volume*

Max\_Capital = Max\_Size \* Average\_Price \* Point\_Value

**Transaction Cost Analysis**

python

*# Total cost per trade*

Cost = Commission + Spread/2 + Slippage + Market\_Impact

*# Break-even win rate*

BE\_win\_rate = (1 + Cost/Avg\_Loss) / (1 + Avg\_Win/Avg\_Loss)

*# Required edge*

Min\_Edge = 2 \* Cost / Average\_Price

Current\_Edge = E[R]

Edge\_Ratio = Current\_Edge / Min\_Edge

Target: Edge\_Ratio > 2.0

**6. Statistical Process Control**

**Performance Monitoring**

python

*# CUSUM chart for returns*

S\_n = max(0, S\_n-1 + (r\_n - μ - k))

where k = tolerance parameter = 0.5 \* σ

*# Alert if S\_n > h (typically h = 4σ)*

*# EWMA control chart*

EWMA\_n = λ \* r\_n + (1-λ) \* EWMA\_n-1

Control\_Limits = μ ± 3σ√(λ/(2-λ))

**Regime Break Detection**

python

*# Chow test for structural break*

RSS\_full = sum of squared residuals (full sample)

RSS\_1 + RSS\_2 = sum for subsamples

F = ((RSS\_full - RSS\_1 - RSS\_2)/k) / ((RSS\_1 + RSS\_2)/(n-2k))

Significant break if F > F\_critical(k, n-2k)

**7. Implementation Validation**

**Execution Quality Metrics**

python

*# Implementation Shortfall*

IS = (Execution\_Price - Decision\_Price) / Decision\_Price

*# VWAP Slippage*

VWAP\_Slippage = (Execution\_Price - VWAP) / VWAP

*# Fill Rate*

Fill\_Rate = Filled\_Orders / Total\_Orders

Target: Fill\_Rate > 0.95

*# Latency Impact*

Latency\_Cost = correlation(latency, slippage)

Target: |correlation| < 0.3

**System Reliability**

python

*# Uptime requirement*

System\_Uptime = Active\_Time / Market\_Hours

Target: > 0.995

*# Order accuracy*

Error\_Rate = Failed\_Orders / Total\_Orders

Target: < 0.001

*# Recovery time*

Mean\_Time\_To\_Recovery < 60 seconds

**8. Final Acceptance Criteria**

**Minimum Statistical Requirements**

python

*# Per strategy:*

1. Sample\_Size > 100 trades

2. Sharpe\_Ratio > 1.0

3. Profit\_Factor > 1.5

4. Win\_Rate \* Avg\_Win > 1.2 \* Loss\_Rate \* Avg\_Loss

5. Max\_Drawdown < 15%

6. P(profit) > 0.95 (Monte Carlo)

7. WFE > 0.5

8. All parameters stable (±20%)

9. Positive in 2/3 market regimes

10. Transaction costs < 30% of gross profit

*# Portfolio level:*

1. Combined\_Sharpe > 1.5

2. Correlation between strategies < 0.5

3. Combined\_Max\_DD < 10%

4. Risk\_of\_Ruin < 0.01